ANL-ART-122

FY2017 Updates to the SAS4A/SASSYS-1 Safety Analysis Code

Nuclear Engineering Division

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (http://www.osti.gov/scitech/)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: orders@ntis.gov

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: reports@osti.gov

ANL-ART-122

FY2017 Updates to the SAS4A/SASSYS-1 Safety Analysis Code

prepared by
T. H. Fanning

Nuclear Engineering Division
Argonne National Laboratory

September 30, 2017

ABSTRACT

The SAS4A/SASSYS-1 safety analysis software is used to perform deterministic analysis of anticipated events as well as design-basis and beyond-design-basis accidents for advanced fast reactors. It plays a central role in the analysis of U.S. DOE conceptual designs, proposed test and demonstration reactors, and in domestic and international collaborations.

This report summarizes the code development activities that have taken place during FY2017. Extensions to the void and cladding reactivity feedback models have been implemented, and Control System capabilities have been improved through a new virtual data acquisition system for plant state variables and an additional Block Signal for a variable lag compensator to represent reactivity feedback for novel shutdown devices.

Current code development and maintenance needs are also summarized in three key areas: software quality assurance, modeling improvements, and maintenance of related tools. With ongoing support, SAS4A/SASSYS-1 can continue to fulfill its growing role in fast reactor safety analysis and help solidify DOE's leadership role in fast reactor safety both domestically and in international collaborations.

TABLE OF CONTENTS

Abstract i
Table of Contents iii
List of Figures iv
List of Tables iv
1 Introduction 1
2 Current Status 1
2.1 Government Use 1
2.2 New License Agreements 1
3 New Features 3
3.1 Higher Order Void Reactivity Feedback 3
3.2 Clad Reactivity Feedback 4
3.3 Plant Virtual Data Acquisition System 5
3.4 Control System Extension 8
4 Other Code Improvements 11
4.1 Documentation 11
4.2 Compliance with Fortran 2003 Standards 12
4.3 Cross-Platform Consistency 12
4.4 Support for 64-bit Compilation 12
4.5 Bug Fixes 12
5 Current Development Needs 13
5.1 Software Quality Assurance 13
5.2 Modeling Improvements 13
5.3 Support for Related Tools 14
6 Summary 15
7 References 15

LIST OF FIGURES

Figure 1: Results of Applying a Variable Lag Compensator to an Incoming Sine Function ... 11

LIST OF TABLES

Table 1: U.S. DOE Research and Development Activities Using SAS4A/SASSYS-1 2
Table 2: SAS License Agreements Established during FY2017 3
Table 3: Explicit `SAS_SYSDASSensor` Subclasses 7
Table 4: Input Requirements for a Variable Lag Compensator 9

1 Introduction

SAS4A/SASSYS-1 is a simulation tool used to perform deterministic analysis of anticipated events as well as design basis and beyond design basis accidents for advanced liquid-metal-cooled nuclear reactors. [1] With its origin as SAS1A in the late 1960s, the SAS series of codes has been under continuous use and development for nearly fifty years. It has been identified as a critical element of safety analysis capabilities for the U.S. Department of Energy. [2,3]

Version 5.2 of SAS4A/SASSYS-1 was completed in March 2017 and released to users in mid-May following a software quality assurance review and publication of the updated manual. [1] Sixteen organizations have requested and obtained the new version. The new release introduces an extension to the Control System module that provides access to an extensive set of core and core channel state variables such as fuel, cladding, coolant, and structure temperatures; coolant flow rates and pressures; and several other parameters. [4]

This report summarizes the code development and update activities carried out during FY2017. Active programs and collaborations that use SAS4A/SASSYS-1 are summarized in Section 2. New features that have been developed since the release of Version 5.2 are summarized in Section 3, and general code improvements are summarized in Section 4. Current development needs are summarized in Section 5.

2 Current Status

2.1 Government Use

Continued maintenance and improvement of the SAS4A/SASSYS-1 code system is motivated by the importance of its simulation capability to a number of U.S. Department of Energy programs as well as domestic and international collaborations. U.S. DOE activities that rely on SAS4A/SASSYS-1 are summarized in Table 1.

2.2 New License Agreements

During FY2017, eight new license agreements were established for SAS4A/SASSYS-1. These are summarized in Table 2. Globally, there are twenty-six licensed users/organizations with access to SAS4A/SASSYS-1 or Mini SAS.

Table 1: U.S. DOE Research and Development Activities Using SAS4A/SASSYS-1

Program/Activity	Description
Test/Demo Reactor (Versatile Test Reactor)	The U.S. DOE has made an assessment of advanced reactor technology options to provide technical context for future decisions concerning irradiation testing capabilities. [5] Safety analyses and licensing activities for a fast spectrum test reactor require the modeling capabilities of SAS4A/SASSYS-1.
FFTF Benchmark	The U.S. DOE is preparing benchmark specifications for the Passive Safety Tests (PST) carried out at the Fast Flux Test Facility between 1984 and 1986. The most prominent tests were the loss of flow without scram (LOFWOS). In collaboration with PNNL, Argonne is assessing the benchmark specifications and preparing SAS4A/SASSYS-1 models for verification and validation purposes. [6]
EBR-II IAEA Benchmark	The U.S. DOE concluded a high-profile Coordinated Research Project with the International Atomic Energy Agency based on the Shutdown Heat Removal Tests conducted at EBR-II. [7] U.S. analyses for both protected (SHRT-17) and unprotected (SHRT-45R) loss-of-flow tests were completed using SAS4A/SASSYS-1.
DOE/CEA Bilateral Collaboration	An implementation agreement has been established between the U.S. DOE and the Commissariat à l'énergie atomique et aux énergies alternatives (CEA) of France for cooperation in low carbon energy technologies. One purpose of the agreement is to evaluate the safety performance of the ASTRID reactor design. DOE participates in this collaboration using the SAS4A/SASSYS-1 safety analysis code.
DOE/JAEA Bilateral Collaboration	The Civil Nuclear Energy Research and Development Working Group (CNWG) was established by the U.S.-Japan Bilateral Commission on Civil Nuclear Cooperation in 2012 to enhance coordination of joint nuclear research and development. The Japan Atomic Energy Agency (JAEA) and Argonne are collaborating under the CNWG to improve the oxide fuel severe accident modeling capabilities in SAS4A/SASSYS-1.
DOE/CIAE Bilateral Collaboration	The DOE-NE Office of International Nuclear Energy Policy and Cooperation has established the U.S.–China Bilateral Civil Nuclear Energy Cooperative Action Plan (BCNECAP) with the China Institute of Atomic Energy (CIAE). Joint activities under the action plan include model development and safety analyses of the China Experimental Fast Reactor using SAS4A/SASSYS-1.
NEUP (University of California—Berkeley)	The University of California at Berkeley is using SAS4A/SASSYS-1 to evaluate safety benefits that might be achieved with autonomous reactivity control devices in sodium-cooled fast reactors.
NEUP (Kansas State University)	Kansas State University is preparing experiments with liquid gallium that can improve the modeling of thermal stratification in SFRs. In collaboration, the University of Illinois will develop improved stratification models that could be incorporated into SAS4A/SASSYS-1.
NEUP (University of Wisconsin)	The University of Wisconsin is preparing experiments with liquid sodium that can improve the modeling of thermal stratification in SFRs. In collaboration, Virginia Commonwealth University is working to develop improved models that could be incorporated into SAS4A/SASSYS-1.

Table 2: SAS License Agreements Established during FY2017

Organization	Code	Purpose
Japan Atomic Energy Agency	SAS4A/SASSYS-1	CNWG Bilateral
U.S. Nuclear Regulatory Commission	SAS4A/SASSYS-1	Government Use
GE-Hitachi Nuclear Energy	Mini SAS	ARC-100 Support
Westinghouse Electric Company	SAS4A/SASSYS-1	LFR Concept
Ulsan National Institute of Science and Technology (UNIST)	SAS4A/SASSYS-1	Academic Use/ PG-SFR
CITON — Center of Technology and Engineering for Nuclear Projects	Mini SAS	EU ALFRED Project
Illinois Rocstar, LLC	SAS4A/SASSYS-1	SBIR Project
Idaho National Laboratory	SAS4A/SASSYS-1	Government Use

3 New Features

New features have been implemented in SAS4A/SASSYS-1 to account for higher-order reactivity feedback effects, to implement a plant-wide virtual data acquisition system, and to update the Control System to implement a variable lag compensator. These developments have been completed, but they have not passed final software quality assurance reviews and will be part of a future release.

3.1 Higher Order Void Reactivity Feedback

The existing reactivity feedback models in SAS4A/SASSYS-1 assume that all feedback mechanisms are linear. Previous studies have shown that for significant sodium voiding, non-linear voiding effects can increase average core outlet temperatures by about 25°C compared to linear feedback assumptions. [8] Recent studies on LFR systems show that the linear feedback assumption is acceptable up to about 10% void. [9] Beyond that, non-linear effects start to become important.

In typical U.S. SFR designs, it is assumed that sodium boiling can be practically eliminated, thus the capability to model non-linear void effects may not be essential. However, fuel pin failures can inject fission gases into a coolant channel [10] and introduce an effect similar to sodium boiling. Since it is straightforward to introduce a non-linear component to the feedback models, it is prudent to do so.

In SAS4A/SASSYS-1, void reactivity feedback is determined from changes in the coolant mass distributions in every channel:

$$\rho_{\text{void}} = \sum_i \sum_j \rho_{\text{void}}^{i,j} \left[m_{\text{cool}}^{i,j}(0) - m_{\text{cool}}^{i,j}(t) \right]$$

where $\rho_{\text{void}}^{i,j}$ is the user-defined void reactivity worth per kilogram (`VOIDRA`) in axial node j of channel i and m_{cool} is the coolant mass. The above equation applies to coolant density changes introduced by thermal expansion, boiling, or fission gas injection.

As shown above, the current model assumes that void reactivity is linear with changes in the coolant density. Non-linear effects are not considered. In the extended model, second-order effects are now considered. For example, if the change in the void mass distribution is defined as

$$\Delta v^{i,j}(t) = m^{i,j}_{\text{cool}}(0) - m^{i,j}_{\text{cool}}(t)$$

then the extended model can be written as

$$\rho_{\text{void}} = \sum_i \sum_j \rho^{i,j}_{\text{void}} \Delta v^{i,j}(t) + \rho^{i,j}_{\text{void,2}} \left[\Delta v^{i,j}(t)\right]^2$$

where $\rho^{i,j}_{\text{void,2}}$ is a new user-defined input (`VOIDRA2`) with units of void reactivity per kilogram-squared. In the absence of new input, the code will assume $\rho^{i,j}_{\text{void,2}} = 0$ and the current model is maintained. The extended implementation currently only considers coolant density changes. The treatment of boiling or fission-gas injection requires additional updates to the severe accident models in SAS4A/SASSYS-1.

To further improve usability, values for void reactivity feedback can now be defined on an arbitrary mesh independent of the SAS4A/SASSYS-1 channel mesh. This provides much-needed flexibility in providing reactivity feedback input from perturbation theory results that may have been computed on a different mesh. The new input is represented by a free-formatted, arbitrary-length, channel-dependent input table:

TABLE <n> "Reactivity"		
LENGTH	VOIDRA	VOIDRA2
Δz_1	$\rho^{i,1}_{\text{void}}$	$\rho^{i,1}_{\text{void,2}}$
Δz_2	$\rho^{i,2}_{\text{void}}$	$\rho^{i,2}_{\text{void,2}}$
…		
Δz_J	$\rho^{i,J}_{\text{void}}$	$\rho^{i,J}_{\text{void,2}}$
END		

The `LENGTH` column is optional. If not provided, SAS4A/SASSYS-1 will assume that the reactivity feedback values correspond to the axial channel mesh. Otherwise, any level of axial detail may be specified for the reactivity feedback coefficients.

3.2 Clad Reactivity Feedback

In SAS4A/SASSYS-1, clad reactivity feedback is determined from changes in the clad mass distributions in every channel:

$$\rho_{\text{clad}} = \sum_i \sum_j \rho^{i,j}_{\text{clad}} \left[m^{i,j}_{\text{clad}}(t) - m^{i,j}_{\text{clad}}(0)\right]$$

where $\rho^{i,j}_{\text{clad}}$ is the user-defined clad reactivity worth per kilogram (`CLADRA`) in axial node j of channel i and m_{clad} is the clad mass. The above equation applies to both fuel-pin axial expansion and clad relocation due to failure.

As shown above, the current model assumes that clad reactivity is linear with changes in the mass distribution. Non-linear effects are not considered. In the extended model, second-order effects are now considered. For example, if the change in the clad mass distribution is defined as

$$\Delta m_{\text{clad}}^{i,j}(t) = m_{\text{clad}}^{i,j}(t) - m_{\text{clad}}^{i,j}(0)$$

then the extended model can be written as

$$\rho_{\text{clad}} = \sum_i \sum_j \rho_{\text{clad}}^{i,j} \Delta m_{\text{clad}}^{i,j}(t) + \rho_{\text{clad},2}^{i,j} \left[\Delta m_{\text{clad}}^{i,j}(t)\right]^2$$

where $\rho_{\text{clad},2}^{i,j}$ is a new user-defined input (`CLADRA2`) with appropriate units. In the absence of new input, the code will assume $\rho_{\text{clad},2}^{i,j} = 0$ and the current model is maintained. The extended model only considers cladding axial expansion. Treatment of clad melting and relocation requires additional updates to the severe accident models.

Like the void reactivity model, the new model allows the values for clad reactivity to be defined on an arbitrary mesh. The new reactivity table can include values for `CLADRA` and `CLADRA2`:

TABLE <n> "Reactivity"				
LENGTH	VOIDRA	VOIDRA2	CLADRA	CLADRA2
Δz_1	…	…	$\rho_{\text{clad}}^{i,1}$	$\rho_{\text{clad},2}^{i,1}$
Δz_2	…	…	$\rho_{\text{clad}}^{i,2}$	$\rho_{\text{clad},2}^{i,2}$
…				
Δz_I	…	…	$\rho_{\text{clad}}^{i,J}$	$\rho_{\text{clad},2}^{i,J}$
END				

The `LENGTH` column is optional. If not provided, SAS4A/SASSYS-1 will assume that the reactivity feedback values correspond to the axial channel mesh.

3.3 Plant Virtual Data Acquisition System

In SAS4A/SASSYS-1, the Control System refers to the module that simulates the response of a series of mathematical and logical operations performed on zero or more Input Signals and Demand Signals, usually with the intent of manipulating one or more Control Signals. TerraPower sponsored work to develop a virtual data acquisition system for core state variables (Input Signals). [4] The resulting new Control System module is built around object-oriented constructs such as class inheritance and polymorphism and is a near complete rewrite of the legacy implementation. This allows the Control System to be written in terms of a generic data acquisition system and extension to new signal inputs is straightforward. In FY2017, the virtual data acquisition system model was extended to include PRIMAR-4 (whole plant) Input Signals such as piping flow rates, coolant and wall temperatures, coolant and cover gas pressures, and pump speeds.

The extension is implemented in the context of the virtual data acquisition system that defines an abstract interface for accessing arbitrary plant state variables. The Control System uses a *builder* object (an extension of the `SAS_BaseDASBuilder` class) to create *sensor* objects (extensions of the `SAS_BaseDASSensor` class). The Control System uses the sensor objects to access PRIMAR-4 state variables. Different modules can use the same builder to create PRIMAR-4 sensors, but the sensors are never maintained in a centralized container. The module that requests the sensor is responsible for retaining a reference to the sensor object, can access its value through an abstract function interface, and is responsible for deallocating the sensor when it is done using it.

The concept of a *DAS Sensor* is encapsulated as the abstract base class `SAS_BaseDASSensor` with the following interface:

DAS Sensor Method	Description
• Init(sInfo,sData,scale,offset)	Designated initializer. Calls InitSelf(sInfo,sData). All parameters are optional.
• GetValue()	Returns scaled value of raw sensor.
• SetScale(scale)	Sets sensor scaling parameter.
• SetOffset(offset)	Sets sensor offset parameter.
+ InitSelf(sInfo,sData) { }	Initializes instance of a specific sensor. All parameters are optional.
+ *RawValue() = 0*	Returns raw sensor value. *Must be overridden in explicit implementations.*

Procedure interfaces preceded by a bullet (•) may not be overridden. Explicit extensions of the abstract base class *must* override the `RawValue` method to implement the code needed to evaluate and return the value corresponding to a specific PRIMAR-4 state variable. The Control System, or other modules that use the sensor, calls `GetValue()` to obtain the current sensor value. This can be done through a reference to the abstract base class, so the Control System never needs to know the explicit implementation.

The inclusion of optional *scale* and *offset* parameters support simple linear transformations of the raw sensor data. If not defined, *scale* = 1 and *offset* = 0. The linear transformation has the following form:

$$\text{GetValue} = \text{scale} \times \text{RawValue} + \text{offset}$$

A simple example of a transformation is one that converts an absolute temperature from Kelvin to Celsius, where *scale* = 1.0 and *offset* = -273.15.

In order to be used, sensors are first allocated and initialized. This is done by a *DAS Builder*, which is represented by the abstract base class `SAS_BaseDASBuilder` with the following interface:

DAS Builder Method	Description
+ NewSensor(sType,sInfo,sData, scale,offset) = 0	Signal construction function. sType is required, other parameters are optional.

Explicit extensions of the abstract base class *must* override the `NewSensor` method to allocate and initialize a new sensor. The explicit implementation uses the `sType` parameter to identify the sensor type to allocate. The remaining parameters are used to initialize the newly allocated sensor by calling `Init(sInfo,sData,scale,offset)` for the sensor. Code modules refer to an explicit builder through a reference to the base class and never need to know how the builder is implemented.

The Control System accesses PRIMAR-4 state variables through the abstract interface described above. Specifically, the **`SAS_SYSDASBuilder`** class extends the `SAS_BaseDASBuilder` abstract class to implement the functionality needed for the `NewSensor` method. The `NewSensor` method, in turn, generates one of many possible **`SAS_SYSDASSensor`** subclasses that reference PRIMAR-4 state variables, where `SAS_SYSDASSensor` subclasses are extensions of the abstract `SAS_BaseDASSensor` class. The subclasses of `SAS_SYSDASSensor` that have been implemented are summarized in Table 3.

Table 3: Explicit `SAS_SYSDASSensor` Subclasses

iSig	Subclass	Description
-74	NormalizedDecayPower	Normalized Decay Power
-73	NormalizedFissionPower	Normalized Fission Power
-72	NormalizedTotalPower	Normalized Total Power
-71	CoreChannelBCTemperature	Core Channel Boundary Condition Temperature
-70	n/a	(undefined)
-69	LiquidElementTemperature	Liquid Element Temperature
-68	TGNodeWallTemperature	Temperature Group Node Wall Temperature
-67	TGNodeLiquidTemperature	Temperature Group Node Liquid Temperature
-66	CoreChannelFlowRate	Core Channel Flow Rate
-65	PumpSpeed	Pump Speed
-64	LSTemperature	Liquid Segment Temperature
-63	UnknownFRNDF3	FRNDF3
-62	CVCoverGasTemperature	Compressible Volume Cover Gas Temperature
-61	CVCoverGasMass	Compressible Volume Cover Gas Mass
-60	CVCoverGasPressure	Compressible Volume Cover Gas Pressure
-59	CVWallTemperature	Compressible Volume Wall Temperature
-58	CVLiquidDensity	Compressible Volume Liquid Density
-57	CVLiquidTemperature	Compressible Volume Liquid Temperature
-56	PumpHead	Pump Head
-55	SubintervalStartTime	Subinterval Start Time
-54	CVCoverGasVolume	Compressible Volume Cover Gas Volume
-53	CVLiquidMass	Compressible Volume Liquid Mass
-52	CVInterfaceElevation	Compressible Volume Interface Elevation
-51	LSFlowRate	Liquid Segment Flow Rate
-50	CVLiquidPressure	Compressible Volume Liquid Pressure

During the transient simulation, the Control System simply calls the `GetValue()` method of each sensor to sample its state. The association between signal numbers (`iSig`) and subclasses is maintained by the parameter array **`SYSDASConfig`** with the help of type identifiers in the module **`SAS_SYSDASTypes`**.

In the past, adding new "sensors" required modifications to several inter-dependent Control System routines and code maintenance was challenging. Now, the concept of a sensor is separate from the implementation of the Control System. If a new PRIMAR-4 sensor needs to be defined, only a few lines of code need to be added:

1. Create a new subclass of `SAS_SYSDASSensor` that references the additional PRIMAR-4 state variable.
2. Update the enumerations in the module `SAS_SYSDASTypes` to identify the new subclass.
3. Update the `NewSensor` method of the `SAS_SYSDASBuilder` class to recognize and allocate the new `SAS_SYSDASSensor` type.

In addition, for the new sensor to be recognized by the control system,

4. Update the parameter array `SYSDASConfig` to associate a new signal number with the new sensor type.

3.4 Control System Extension

As described above, the Control System performs a series of mathematical and logical operations on zero or more Input Signals and Demand Signals with the intent of manipulating one or more Control Signals. In response to needs identified by a Nuclear Energy University Partnership project, new mathematical operations were required to represent the reactivity feedback characteristics of an Autonomous Reactivity Control device. [11]

The behavior of an ARC device can be represented by a *variable lag compensator*. A variable lag compensator is similar to a lag compensator except that the delay time parameter can vary during the Control System simulation. An equation that expresses this behavior is

$$y(t) + \tau(t)\frac{dy}{dt} = u(t)$$

where

$u(t)$ = input driving function

$\tau(t)$ = delay (lag)

$y(t)$ = output function result

Assuming that the lag time is always positive, a solution to the above equation is

$$y(t) = y_0 e^{-\int_0^t \frac{dt'}{\tau(t)}} + e^{-\int_{t_0}^t \frac{dt'}{\tau(t)}} \int_0^t \frac{u(t')}{\tau(t')} e^{\int_{t_0}^{t'} \frac{dt''}{\tau(t'')}} dt'$$

where $y_0 = y(0)$ and t_0 is an arbitrary integration constant.

The complexity of the above solution can be simplified if we assume the input driving function and delay function are linear functions of time: $u(t) = u_0 + u't$ and $\tau(t) = \tau_0 + \tau' t$, respectively. With this assumption, the solution becomes

$$y(t) = y_0\left(1 + \frac{\tau'}{\tau_0}t\right)^{-1/\tau'} + u_0\left[1 - \left(1 + \frac{\tau'}{\tau_0}t\right)^{-1/\tau'}\right] + \frac{u'}{1+\tau'}\left[t - \tau_0\left[1 - \left(1 + \frac{\tau'}{\tau_0}t\right)^{-1/\tau'}\right]\right]$$

In the limit that $\tau' \to 0$, we arrive at the solution for constant τ during the time interval:

$$y(t) = y_0 e^{-t/\tau_0} + u_0\left(1 - e^{-t/\tau_0}\right) + u'\left[t - \tau_0\left(1 - e^{-t/\tau_0}\right)\right]$$

The analytic solution for the variable lag compensator can be used to solve for the value of $y_1 = y(t_0 + \Delta t)$ at the end of a Control System sub-interval (time step) given the initial value $y_0 = y(t_0)$. Knowing the lag time at the beginning of the time step, τ_0, and the input driving functions at the beginning and end of the time step, $u_0 = u(t_0)$ and $u_1 = u(t_0 + \Delta t)$, respectively, then the solution for y_1 at the end of the time step is

$$y_1 = y_0 e^{-\delta} + u_0\left(1 - e^{-\delta}\right) + (u_1 - u_0)\left[1 - \frac{1 - e^{-\delta}}{\delta}\right]$$

where $\delta = \Delta t/\tau_0$ is a dimensionless time step size. Because this solution is prone to numerical round-off errors, expansions for the exponential function are used:

$$e^{-\delta} = 1 - \delta + \frac{\delta^2}{2} - \frac{\delta^3}{6} + O[\delta^4]$$

and

$$\frac{1 - e^{-\delta}}{\delta} = 1 - \frac{\delta}{2} + \frac{\delta^2}{6} - \frac{\delta^3}{24} + O[\delta^4]$$

Using these expansions, the solution is written as

$$y_1 = y_0 \left[1 - \delta + \frac{\delta^2}{2}\right] + u_0 \left[\frac{\delta}{2} - \frac{\delta^2}{3}\right] + u_1 \left[\frac{\delta}{2} - \frac{\delta^2}{6}\right] + O[\delta^3]$$

To minimize numerical round-off errors in the code, the following form is used:

$$y_1 \approx y_0 \left[1 - \delta\left(1 - \delta\left(\frac{1}{2}\right)\right)\right] + u_0 \left[\delta\left(\frac{1}{2} - \delta\left(\frac{1}{3}\right)\right)\right] + u_1 \left[\delta\left(\frac{1}{2} - \delta\left(\frac{1}{6}\right)\right)\right]$$

which is second-order accurate in δ. This form also avoids the need to make expensive exponential function evaluations.

A variable lag compensator is identified in the Control System input as signal type `ITYPE = 23`. It requires two input signals to define the time-dependent functions $u(t)$ and $\tau(t)$. These are identified by the input parameters `J1` and `J2`, respectively. Input for the variable lag compensator is summarized in Table 4.

Table 4: Input Requirements for a Variable Lag Compensator

Input	Description
ISIG	User-defined signal number
ITYPE = 23	Variable Lag Compensator type
J1	Signal # of driving function, $u(t)$
J2	Signal # of delay time, $\tau(t)$
F1	Multiplier for driving function default: 1
F2	(unused)
F3	(unused)
F4	Steady-state initial condition, $y(0)$ default: $y(0) = u(0)$
F5	Zero-crossing parameter (`F5 > 0`)

Additional input includes two optional parameters, `F1` and `F4`, and one required parameter, `F5`. If the optional parameters are blank or zero, defaults will be applied as indicated in the table. For example, if `F1` is non-zero, then the actual driving function will include a multiplier:

$$u(t) = F1 \times \mathrm{sig}(J1)$$

Likewise, if F4 is non-zero, it is used to define the initial steady-state condition as $y(0) = F4$. Otherwise, the steady-state solution to the equation is $y(0) = u(0)$.

The zero-crossing parameter, F5, is used in the same manner as with other Control System signals to stabilize convergence testing when the solution for $y(t)$ is near zero. The normalized error is approximated as follows to avoid divide-by-zero errors:

$$\text{error} = \left|\frac{\Delta y}{y}\right| \cong \frac{|\Delta y|}{|y| + F5}$$

Therefore, users can select a value for F5 that is small enough to ensure convergence when $y(t)$ is small, but large enough to avoid excessive iterations when small values for $y(t)$ do not have a significant impact on the overall solution.

Like the lag compensator, the variable lag compensator shifts both the phase and amplitude of the input signal $u(t)$. However, the phase and amplitude vary as a function of time through the additional input for $\tau(t)$. To demonstrate this, consider the following conditions:

$$u(t) = \sin\left(\frac{2\pi}{5}t\right)$$
$$\tau(t) = 1.9\sin\left(\frac{2\pi}{50}t\right) + 2$$
$$y(0) = 2$$

These can be represented in the Control System input as follows:

```
INCONT     5
# Time: t
    1  -55    0    0       0.0       0.0       0.0       0.0       0.0
# Driving Signal: Sin(2πt/5)
   11   22    1    0       1.0 1.2566370       0.0       0.0
# Lag Time Variation: Sin(2πt/50)
   12   22    1    0       1.0 .12566370       0.0       0.0
# Constant: 2
   13   21    0    0       2.0       0.0       0.0       0.0       0.0
# Lag Time: 1.9Sin(2πt/50) + 2
   17    1   12   13       1.9       1.0       1.0       0.0       0.0
# Variable Lag Compensator(11,17)
#     ITYPE  J1   J2        F1                             F4        F5
   18   23   11   17       1.0       0.0       0.0       2.0     0.001
  999    1
  8001     2  0.00000001  0.00000001
END
```

Using the above Listing in a SAS4A/SASSYS-1 input file would produce the Control System output shown in Figure 1.

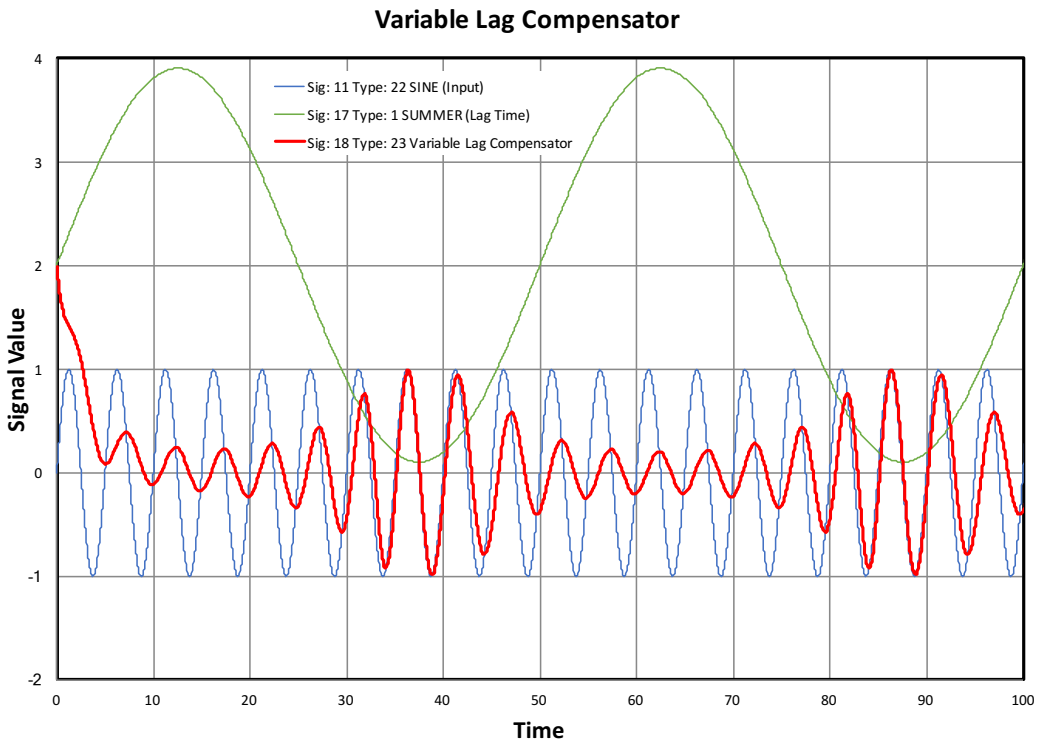

Figure 1: Results of Applying a Variable Lag Compensator to an Incoming Sine Function

Note that because `F4 = 2`, the variable lag compensator starts at the non-equilibrium value $y(0) = 2$. As the lag time $\tau(t)$ rises and falls, the delay in the input signal becomes longer and shorter, respectively. Similarly, the amplitude of the output decreases and increases.

4 Other Code Improvements

Numerous additional code improvements have been made during FY2017. These are summarized briefly in the following sections.

4.1 Documentation

With the release of SAS4A/SASSYS-1 version 5.2, the code manual has also been updated. The latest release is

> T. H. Fanning, A. J. Brunett, and T. Sumner, eds., *The SAS4A/SASSYS-1 Safety Analysis Code System*, ANL/NE-16/19, Nuclear Engineering Division, Argonne National Laboratory, March 31, 2017.

The manual consists of multiple documents spanning 2200+ pages that detail user input, theoretical considerations, and implementation details. The sizes of the documents have become unmanageable as traditional office-type documents. A small activity has been initiated to convert the documents to a markup format that is more amenable to frequent updates and edits that can be tracked in an SQA environment.

4.2 Compliance with Fortran 2003 Standards

Although the history of SAS4A/SASSYS-1 spans nearly five decades, it has undergone continuous updates to maintain compliance with software programming standards. The most recent update brings the source code into compliance with Fortran 2003 standards. Obsolete code that has been eliminated includes

- 73 arithmetic "`if`" statements
- 181 computed "`go to`" statements
- 4,141 DO loops without proper "`end do`" termination
- 350 invalid edit descriptors

Non-standard code that will continue to be accepted includes the use of tab characters (for indenting) and source code line lengths that extend beyond 132 characters. These extensions support code refactoring as the source undergoes continued modernization.

4.3 Cross-Platform Consistency

Code testing and validation is always a challenge. This challenge is made even more difficult when software is maintained for multiple platforms, as is the case with SAS4A/SASSYS-1 which supports macOS, Linux, and Windows systems. With careful tuning of compiler options, SAS4A/SASSYS-1 now produces bit-identical results on all three platforms when using the Intel Fortran compiler. This is true even for optimized code. Testing on all three platforms now uses the same set of reference solutions, which significantly increases the value of regression testing.

4.4 Support for 64-bit Compilation

SAS4A/SASSYS-1 can now be compiled either as a 32-bit or 64-bit application. Compiling as a 32-bit executable is still the default because it is required when coupling with the spatial kinetics capabilities of DIF3D-K. Compiling as a 64-bit application allows for larger problem domains and memory usage, although it is not a critical need. The main motivation for compiling in 64-bit was to identify and eliminate memory access bugs. Both 32- and 64-bit versions now produce bit-identical results and can be tested against the same reference solutions.

One issue remains, however. In extensive testing, a round-off error was identified in the Intel Fortran *libraries*. The problem appears in the 32-bit I/O libraries that ship with the Linux and macOS distributions for Intel Fortran. It does not appear in the 32-bit libraries for Windows or the 64-bit libraries for any platform. This issue has been reported to Intel.

4.5 Bug Fixes

A number of issues have been identified and corrected since the release of SAS4A/SASSYS-1 Version 5.2. These corrections will be incorporated into the forthcoming 5.2.1 revision. Corrected issues include the following:

- `RESTART` files are no longer generated if `NSTEP = 0`, consistent with documentation.
- Eliminated a potential divide-by-zero error when the Young's Modulus for cladding and fuel are not provided in input.
- Corrected the declaration of a local variable used in debug print statements.
- Corrected an issue where the input parameter `IPRION` was not treated consistently when PRIMAR-4 was not being used.

- Eliminated a potential floating-point exception on Windows when debug prints are enabled.
- Eliminated a potential divide-by-zero error when predicting the time-step cutback in PRIMAR-4 if core channel flow rates are not changing.
- Corrected an issue where a non-zero value for `IFLOW` would impact PRIMAR-4 calculations.
- Eliminated a rare divide-by-zero error on Windows caused by poor CPU timing resolution.

5 Current Development Needs

Development needs have been identified in three broad areas. First, the software quality assurance program must be maintained in order for SAS4A/SASSYS-1 to remain viable as a safety analysis tool for nuclear facilities. Second, gaps in modeling capabilities exist that should be addressed on a prioritized basis. Third, SAS4A/SASSYS-1 relies on input from related design and analysis tools. Some of these tools are only maintained on an *ad-hoc* basis.

5.1 Software Quality Assurance

Code maintenance activities have shifted into a more formal software quality assurance Program that has been developed under the Regulatory Technology Development Plan. [12,13] Source code and documentation are maintained within a software configuration management system and formal code changes are tracked in an electronic ticket system according to SQA procedures.

These activities are vital for any software that will be used to assess the safety of a nuclear installation, but they do require resources to maintain. Feedback from other DOE, NRC, and industrial organizations suggest that SQA Program compliance requires between 30–60% of software maintenance resources. The goal of the SAS4A/SASSYS-1 SQA Program is to minimize, to the extent possible, this overhead. For example, the current Program does not intend to achieve NQA-1 compliance [14,15] as this may be better addressed by a commercial-grade dedication process pursued by domestic industry. On the other hand, if DOE intends to authorize construction of a fast spectrum test reactor, then safety analysis codes must comply with DOE quality assurance requirements set forth in DOE Order 414.1D.

In addition to Program maintenance and compliance, additional resources are needed to improve verification and regression testing, code coverage analyses, validation, and documentation. An evaluation of the current test suite reveals that there are important code-coverage gaps that need to be addressed even though the regression test suite has expanded considerably in the past year. Translation of the code documentation into a more manageable file format is also taking place, but at a slow pace.

Adherence to an SQA program will help ensure that software development and documentation activities are carried out in a way to ensure that SAS4A/SASSYS-1 can be used for potential license or authorization of a fast spectrum reactor.

5.2 Modeling Improvements

Improvements to SAS4A/SASSYS-1 not only includes the implementation of new capabilities required to support safety analyses, but source code changes required to improve usability, performance, scalability, and maintenance. The latter is an ongoing process that is needed to

maintain existing capabilities as programming languages and computational resources evolve. The former is based on prioritization of identified gaps in modeling capabilities. Some of these gaps, based on current R&D activities, include the following:

Metal Fuel Transient Modeling: The Korea Atomic Energy Research Institute (KAERI) has invested considerable resources in developing metallic fuel accident modeling capabilities in the context of SAS4A/SASSYS-1. These models are directly relevant to U.S. SFR designs. Currently, however, these models exist in a separate branch of code development. Considerable effort will be needed to reintegrate the updates into the official SAS4A/SASSYS-1 code.

Oxide Fuel Transient Modeling: International organizations, such as CEA and JAEA, have interest in oxide fuel severe accident modeling. SAS4A/SASSYS-1 already has a strong oxide fuel modeling capability, and JAEA is working with Argonne to integrate updates to those models into SAS4A/SASSYS-1. It is important this work continues so that DOE maintains its leadership role in international fast reactor safety.

Integration of Higher-Order Feedback Models: The feedback models described in Section 3.1 were designed to simplify the integration of the metal fuel transient models being developed by KAERI. The feedback models need to be integrated into the severe accident models in SAS4A/SASSYS-1.

Integration with NEAMS Tools: A number of developments being pursued by the Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program would be of benefit to SAS4A/SASSYS-1. Efforts to integrate the System Analysis Module (SAM) should continue, and the potential for NEAMS Workbench to improve usability and to simplify preparation of user input should be evaluated.

5.3 Support for Related Tools

Although SAS4A/SASSYS-1 is a critical component for fast reactor safety analysis, it is not the only software tool required for design, safety analysis, and licensing. In 1977 the U.S. DOE published a "compendium" of computer codes for the safety analysis of fast reactors. [16] At the time, more than 130 codes were identified, although many were R&D tools that may not be essential for authorization or licensing. Forty years since its publication, most of the identified codes no longer exist.

It is beyond the scope of this report to identify all of the computer codes that may be required for fast reactor design and analysis. Within the scope of design-basis and beyond-design-basis safety analysis, however, key tools are required to provide the following:

- Fast spectrum cross-section processing
- Flux and power distributions
- Gamma transport and heating
- Fuel cycle and depletion analysis
- Flow orifice design and optimization
- Kinetics and perturbation theory analysis
- Source term and radionuclide release

Computer codes that provide these capabilities exist, but support and maintenance is *ad-hoc* in most cases. Formal support and SQA programs need to be established for these and other tools required for design and safety analysis.

6 Summary

Over the last few years, the SAS4A/SASSYS-1 safety analysis code has undergone significant revisions to modernize code structure and data management. [17,18,19,4] With continued interest in advanced non-LWR reactors, there is also growing interest in applying SAS4A/SASSYS-1 to a number of SFR and LFR concepts. In total, there are twenty-six license agreements in place, with eight of them established in the current fiscal year. The safety analysis code continues to be of significant importance to DOE sponsored activities as well as to domestic and international collaborations. SAS4A/SASSYS-1 Version 5.2 was completed in March 2017 and released in May.

Development activities that have been completed include extensions to the void and cladding reactivity feedback models, improved Control System capabilities through a new virtual data acquisition system for plant state variables, and an additional Block Signal for a variable lag compensator to represent reactivity feedback for novel shutdown devices.

Current code development and maintenance needs are also summarized in three key areas: software quality assurance, modeling improvements, and maintenance of related tools. With ongoing support, SAS4A/SASSYS-1 can continue to fulfill its growing role in fast reactor safety analysis and help solidify DOE's leadership role in fast reactor safety both domestically and in international collaborations.

7 Acknowledgements

Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Nuclear Energy, under contract DE-AC02-06CH11357.

8 References

1. T. H. Fanning, A. Brunett, and T. Sumner, eds., *The SAS4A/SASSYS-1 Safety Analysis Code System*, ANL/NE-16/19, Nuclear Engineering Division, Argonne National Laboratory, March 31, 2017.
2. M. Denman et al., "Sodium Fast Reactor Safety and Licensing Research Plan – Volume I," SAND2012-4260, Sandia National Laboratories, 2012.
3. Idaho National Laboratory, "Advanced Reactor Technology – Reactor Technology Development Plan (RTDP)," INL/EXT-14-32837, 2015.
4. T. H. Fanning, A. J. Brunett, and G. Zhang, unpublished information, Argonne National Laboratory, September 30, 2016.
5. D. Petti, R. N. Hill, J. Gehin, et al., *Advanced Demonstration and Test Reactor Options Study*, INL/EXT-16-37867, Revision 3, January 2017.
6. T. Sumner, A. Moisseytsev, and F. Heidet, unpublished information, Argonne National Laboratory, September 30, 2017.
7. International Atomic Energy Agency, *Benchmark Analysis of EBR-II Shutdown Heat Removal Tests*, IAEA-TECDOC-1819, Vienna, Austria, August 2017.
8. T. Takada, T. Kuroishi, M. Ohashi, and K. Kaneto, "Neutronic Decoupling and Nonlinearity of Sodium Void Worth of an Axially Heterogeneous LMFBR in ATWS

Analysis," *Proc. Int'l Conf. on Design and Safety of Advanced Nuclear Power Plants*, 25–29 October, 1992.

9. M. Aufiero, M. Martin, M. Fratoni, E. Fridman, and S. Lorenzi, "Analysis of the Coolant Density Reactivity Coefficient in LFRs and SFRs via Monte Carlo Perturbation/Sensitivity," *Proc. PHYSOR 2016*, Sun Valley, ID, May 1–5, 2016.
10. A. Tentner, S. Kang, and A. Karahan, "Advances in the Development of the SAS4A Code Metallic Fuel Models for the Analysis of Prototype Gen-IV Sodium-cooled Fast Reactor Postulated Severe Accidents," *Proc. Int'l Conference on Fast Reactors and Related Fuel Cycles*, FR17, Yekaterinburg, Russia, 26–29 June 2017.
11. S. A. Qvist, "Tailoring the response of Autonomous Reactivity Control (ARC) systems," *Annals of Nuclear Energy*, **99**, pp. 383–398, January 2017.
12. A. J. Brunett, L. L. Briggs, T. H. Fanning, *Status of SFR Codes and Methods QA Implementation*, ANL-ART-83, Argonne National Laboratory, January 31, 2017.
13. A. Brunett and T. H. Fanning, *Implementation of Software QA for SAS4A/SASSYS-1*, ANL-ART-110 Rev. 0, Argonne National Laboratory, September 8, 2017.
14. American Society of Mechanical Engineers, *Quality Assurance Requirements for Nuclear Facility Applications*, ASME NQA-1-2008, 2008.
15. American Society of Mechanical Engineers, *Addenda to ASME NQA-1-2008: Quality Assurance Requirements for Nuclear Facility Applications*, NQA-1a-2009, 2009.
16. U.S. Department of Energy, A Compendium of Computer Codes for the Safety Analysis of Fast Breeder Reactors, DOE/ET-0009, U.S. DOE Division of Reactor Research and Development, October 1977.
17. T. H. Fanning, F. E. Dunn, D. Grabaskas, T. Sumner, and J. W. Thomas, unpublished information, Argonne National Laboratory, September 20, 2013.
18. T. H. Fanning, A. Brunett, T. Sumner, and N. Stauff, unpublished information, Argonne National Laboratory, September 26, 2014.
19. T.H. Fanning, A. Brunett, T. Sumner, R. Hu, unpublished information, Argonne, IL, September 30, 2015.

Nuclear Engineering Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 208
Argonne, IL 60439-4842

www.anl.gov